Examining ASP.NET's Membership, Roles, and Profile - Part 5

By Scott Mitchell

Introduction

ASP.NET 2.0 makes it quite easy to accomplish common tasks. Want to display data from a database, allowing the user to sort, edit, delete, and page through that data? Simply add and configure a SqlDataSource on the page, bind it to a GridView, check a few checkboxes in the GridView's smart tag, and, voila, you have a web-based data entry form that can be created in 15 minutes or so. While the simple case is often a cinch to implement, in the real world rarely is the simple case a practical solution. Often, the simple case needs to be extended and expanded and customized to fit custom rules, logic, formatting, and behavior. Thankfully, ASP.NET 2.0 was designed with extensibility in mind and, thanks to things like the provider model, event handlers, and templates, customizing and extending the simple case is both doable and usually doable without an inordinate amount of effort or "hackery."

As we've seen throughout this article series, ASP.NET provides a platform for creating and managing user accounts through its membership, roles, and profile systems. The related Web controls - Login, LoginView, CreateUserWizard, LoginStatus, and so on - can be used to achieve the simple case. Need to provide an interface for logging on a user? Simply drop the Login Web control onto a page. But what if we want to customize the login experience? We may want to reposition the Web controls used by the Login control or add additional content or Web controls to the Login control interface. Or we may want to customize the credentials supplied by the user for authentication purposes. Rather than requiring just their username and password, what if we want to also make them supply their email address on file? Or perhaps we want to include a CAPTCHA (those boxes with text in an image designed to defeat robot programs from successfully submitting a form).

The Login Web control can be customized in a number of ways. First, it has a bevy of properties that can adjust whether or not the "Remember me next time" checkbox is displayed, the text displayed for the "Log In" Button, the colors, fonts, and other style-related settings, and so on. For further control over the layout of the Login control or of the actual controls that makeup the Login control, we can convert the control into a template. And finally, the control's authentication logic can be customized by creating an event handler for the `Authenticate` event (which can allow us, for example, to use a CAPTCHA as part of the authentication process).

In this article we'll examine how to customize the Login control through its properties, through templates, and by performing custom authentication through an `Authentication` event handler (including an example with a CAPTCHA). Read on to learn more!

Customizing the Login Control Through Its Properties

Before we look at how to alter the Login control's layout or how to customize the authentication process, let's first spend a moment exploring how to customize the standard, "simple case" Login control. The Login control, as you likely know, provides an interface for a user to enter their credentials in order to log on to an ASP.NET 2.0 site that uses the membership system. Therefore, for the examples we'll examine here (which can be downloaded at the end of this article), I'm assuming that you have already configured your website to utilize a membership provider. The demos at the end of this article use the default membership provider, `SqlMembershipProvider`; see Part 1 of this article series for a more thorough discussion at membership's providers and configuring your website to use the default provider.

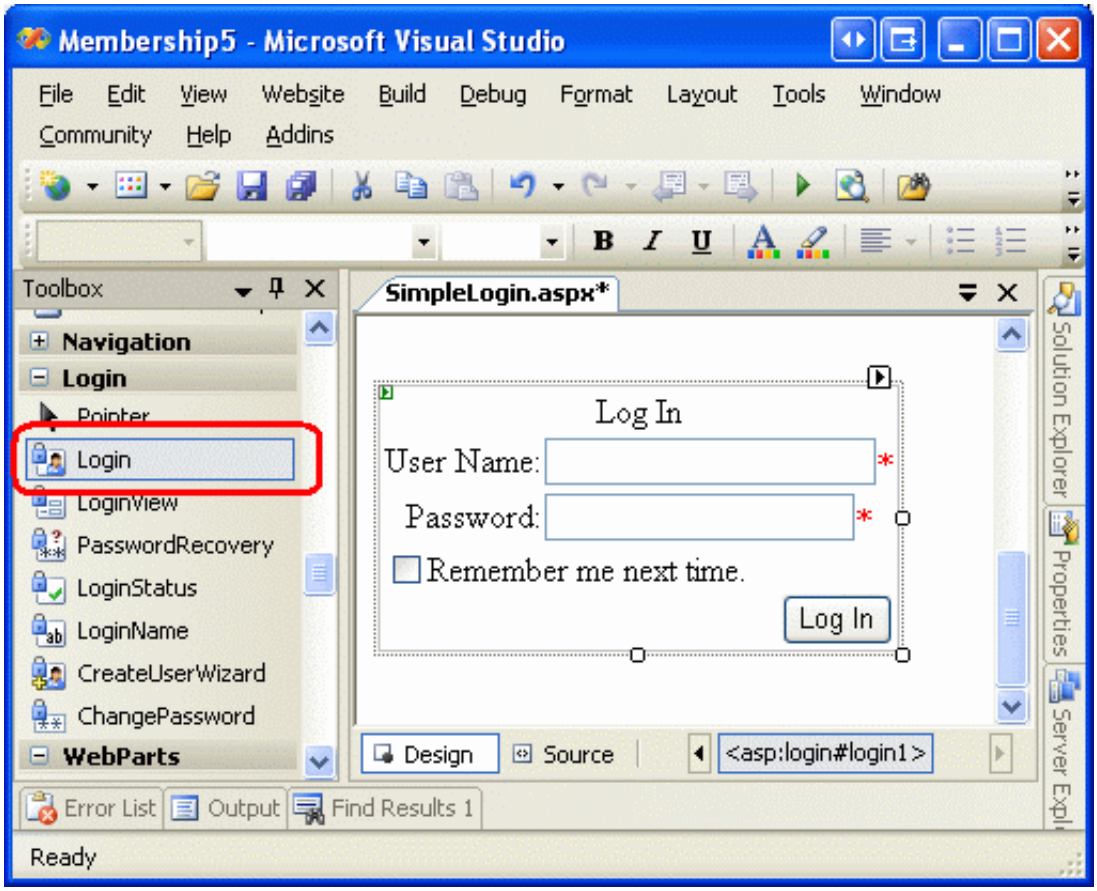
To get started, we need to create a login page. This is the page that users will be automatically redirected to if they attempt to visit a page they are not authorized to view or if they click the Login button from the LoginStatus control. By default, the login page must be named `Login.aspx` and placed in the web application's root directory. You can use an alternative login URL if you like, but you'll need to update the `<forms>` element in the `<authentication>` section in `Web.config` accordingly:

```
<?xml version="1.0"?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
   <system.web>
      ...

      <authentication mode="Forms">
         <forms loginUrl="LoginPath" />
      </authentication>

      ...
   </system.web>
</configuration>
```

To use the "simple case", simply drag the Login control from the Toolbox onto the login page you've created. At this point we have a fully functional login page, albeit a rather unattractive one. As the sceen shot below shows, the Login control, by default, includes:

- A User Name TextBox
- A Password TextBox
- A "Remember me next time" CheckBox
- A "Log In" Button

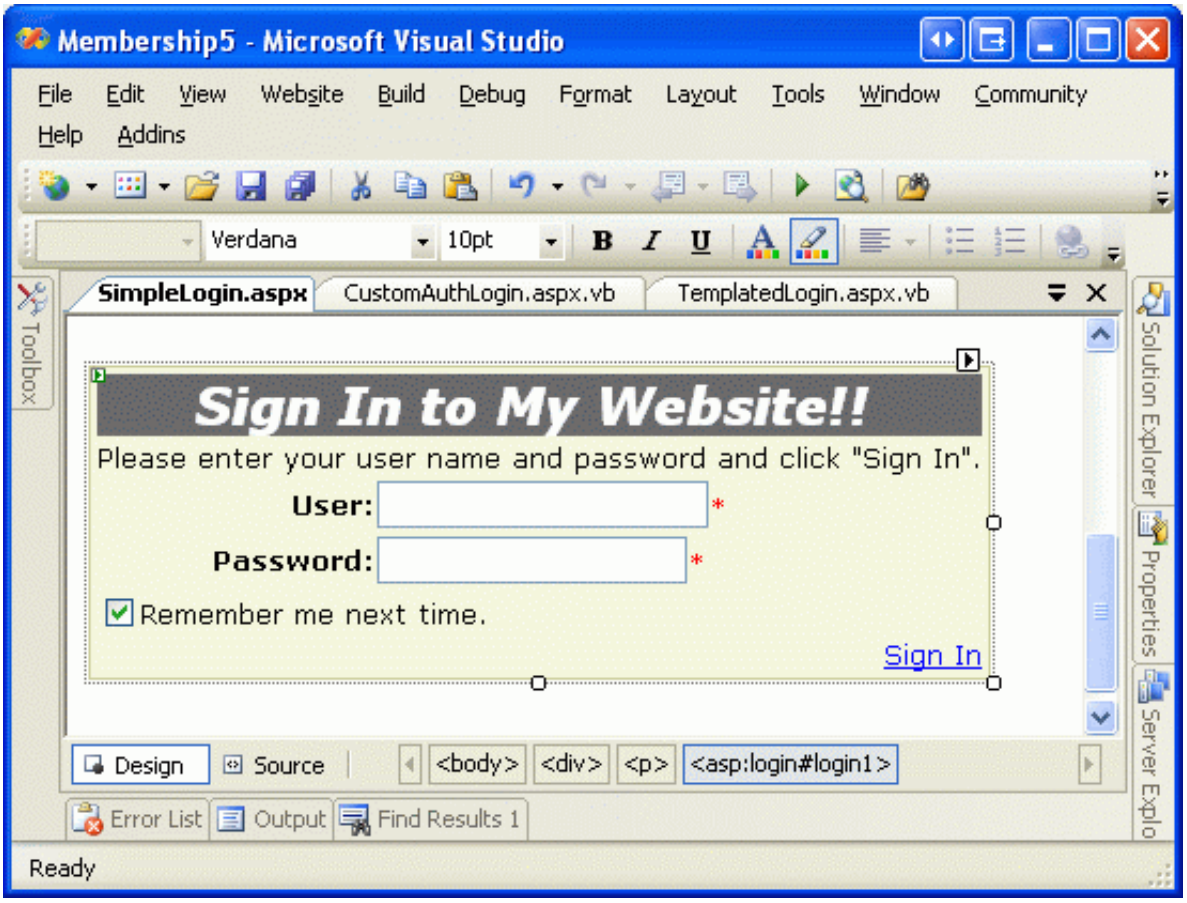
Additionally, the Login control injects RequiredFieldValidators that operate on both the User Name and Password TextBoxes, to ensure that the user has entered a value. Additionally, there's a Literal control that is used to display the Login control's `FailureText` property value in the event of an invalid login.



The Login control's appearance can be touched up through a variety of style-related properties, including control-wide settings such as `Font`, `BackColor`, `Width`, `Height`, and so on, along with style settings specific to certain regions of the control, such as `TextBoxStyle`, `CheckBoxStyle`, and `LoginButtonStyle`, which affect the appearance of the TextBoxes, CheckBox, and Button Web controls in the Login control. For artistically challenged folks like yours truly, there's always the Auto Format option in the control's smart tag!

In addition to the control's appearance, it's properties allow for changes to the text and layout. Want to replace the text "User Name:" with something else? Simply set the `UserNameLabelText` property. You can add instructions to the output by specifying them in the `InstructionText` property; change the text of the "Log In" Button through the `LoginButtonText` property; add a link to a Help page through the `HelpPageText` and `HelpPageUrl` properties. The `Orientation` property allows you to dictate whether the Login controls should be laid out ertically (the default) or horizontally.

The Login control shown below - and avaiable from the download at the end of this article - illustrates how dramatically the appearance can change just through adjusting some property values.



Gaining Complete Control Over the Login Control's Layout

While the Login control's properties allow for a fair degree of customization, in order to have complete control over the layout of the Login control we must convert it into a template. To accomplish this, go to the Designer, expand the Login control's smart tag, and click the "Convert to Template" option. This will add a `<LayoutTemplate>` to the Login control's declarative markup that contains an HTML `<table>` whose structure replicates the default Login control's layout. You can adjust the layout however you like, including adding additional content and Web controls.

One import fact to keep in mind, though, is that certain Login control Web controls *must* be left in the template with their reserved `ID` values. In particular, it's imperative that the `<LayoutTemplate>` include a controls with `ID` values `UserName` and `Password`, and that these controls implement the `ITextControl` interface (which defines that the control has a `Text` property). The TextBox Web control implements `ITextControl`, so you can use those. Alternatively you could use your own custom server controls or User Controls as long as they, too, implemented this interface.

Additionally, you need a Button, LinkButton, or ImageButton whose `CommandName` is set to the value of the Login control's `LoginButtonCommandName` static field (the default value being "`Login`"). But the `ID` of that Button, LinkButton, or ImageButton can be any value. All of the other controls generated by converting the Login control to a template are optional.

The following declarative markup shows how I've adjusted the Login control's `<LayoutTemplate>` to provide a different layout:

```
<asp:Login ID="Login1" runat="server" BackColor="#F7F7DE" BorderColor="#CCCC99"
    BorderStyle="Solid" BorderWidth="1px" Font-Names="Verdana" Font-Size="10pt"
RememberMeSet="True">
    <TitleTextStyle BackColor="#6B696B" Font-Bold="True" ForeColor="White" />
    <LayoutTemplate>
```

```
        <table border="0" cellpadding="1" cellspacing="0" style="border-collapse:
collapse">
            <tr>
                <td align="center" rowspan="3" style="padding:15px; font-weight: bold;
color: white; background-color: #6b696b">
                   Log In
                </td>
                <td style="width:8px;"> </td>
                <td align="right">
                   <asp:Label ID="UserNameLabel" runat="server"
AssociatedControlID="UserName">User Name:</asp:Label></td>
                <td>
                   <asp:TextBox ID="UserName" runat="server" TabIndex="1"></asp:TextBox>
                   <asp:RequiredFieldValidator ID="UserNameRequired" runat="server"
ControlToValidate="UserName"
                        ErrorMessage="User Name is required." ToolTip="User Name is
required." ValidationGroup="Login1">*</asp:RequiredFieldValidator>
                </td>
                <td align="center" rowspan="3" style="padding:15px;">
                    <asp:Button ID="LoginButton" runat="server" CommandName="Login"
Text="Log In" ValidationGroup="Login1" TabIndex="4" />
                </td>
            </tr>
            <tr>
                <td style="width:8px;"> </td>
                <td align="right">
                    <asp:Label ID="PasswordLabel" runat="server"
AssociatedControlID="Password">Password:</asp:Label></td>
                <td>
                    <asp:TextBox ID="Password" runat="server" TextMode="Password"
TabIndex="2"></asp:TextBox>
                    <asp:RequiredFieldValidator ID="PasswordRequired" runat="server"
ControlToValidate="Password"
                        ErrorMessage="Password is required." ToolTip="Password is
required." ValidationGroup="Login1">*</asp:RequiredFieldValidator>
                </td>
            </tr>
            <tr>
                <td style="width:8px;"> </td>
                <td colspan="2">
                    <asp:CheckBox ID="RememberMe" runat="server" Text="Remember me next
time." TabIndex="3" />
                </td>
            </tr>
        </table>
        <asp:ValidationSummary ID="ValidationSummary1" runat="server"
ShowMessageBox="True" ValidationGroup="Login1"
                ShowSummary="False" />
    </LayoutTemplate>
</asp:Login>
```

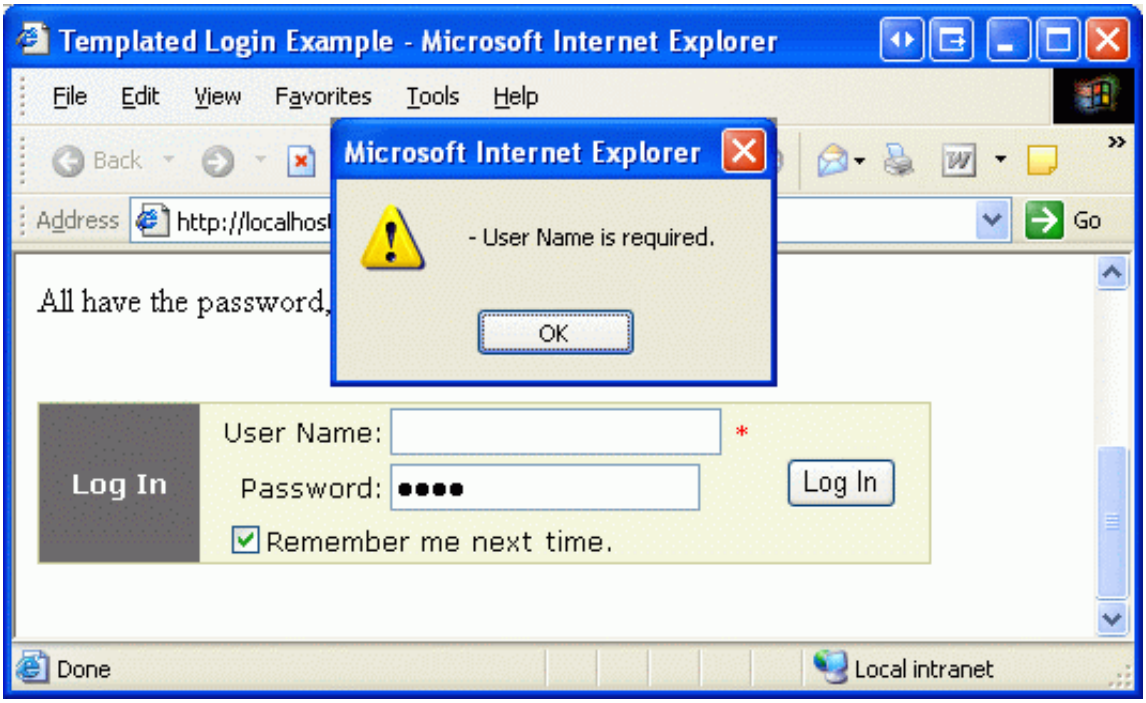
The keen reader will note that the above layout is missing the `FailureText` Literal control. This Literal control displays the value of the Login control's `FailureText` property if there's a problem authenticating the user. Rather than display text in a Literal control, I decided I wanted to show the failure text in a client-side alert box. To accomplish this, I created an event handler for the Login control's `LoginError` event and emitted the JavaScript necessary to display the alert box (we discussed this event and how to

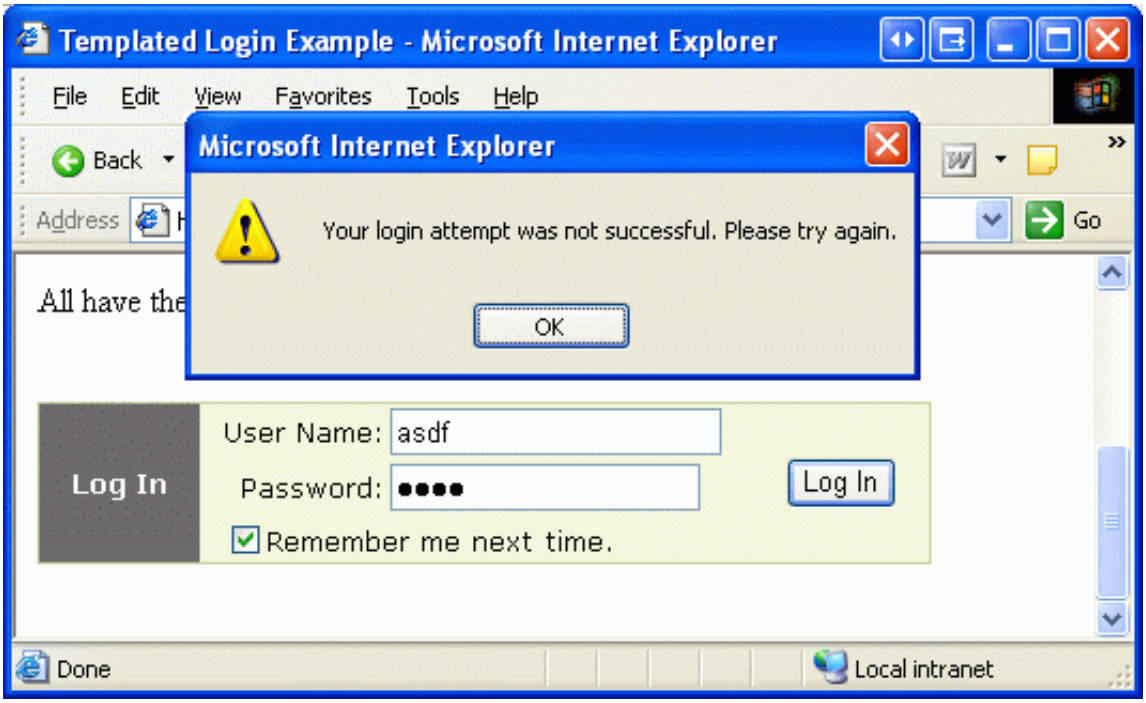
use it to log erroneous login attempts in Part 4 of this article series):

```
Protected Sub Login1_LoginError(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Login1.LoginError
   'Display the failure message in a client-side alert box
   ClientScript.RegisterStartupScript(Me.GetType(), "LoginError", _
               String.Format("alert('{0}');", Login1.FailureText.Replace("'", "\'")),
True)
End Sub
```

Finally, note the inclusion of the ValidationSummary control after the closing `<table>` tag (it's in *italics*). I added this and set its `ShowSummary` and `ShowMessageBox` properties such that it would display an client-side alert box in the face of invalid input (namely if the user does not enter their username and/or password).

The two screen shots below illustrate the customized Login control in action. In the first screen shot, the user sees an alert box explaining that they must provide a username; in the second, the user has attmepted to login, but has entered invalid credentials.





Customizing Authentication Logic

When a user enters their credentials and clicks the "Log In" Button the following steps unfold:

1. A postback ensues
2. The Login control's `LoggingIn` event fires
3. The Login control's `Authenticate` event fires; this invokes an event handler that performs the logic necessary to determine whether the user's credentials are valid. If they are, this event handler must set the `Authenticated` property of the passed-in `AuthenticateEventArgs` object to True
4. If there was a problem authenticating the user, the `LoginError` event is raised; if the user was successfully logged in, the `LoggedIn` event is fired instead

By default, the `Authenticate` event is handled internally by the Login control using the `Membership` class's `ValidateUser(username, password)` method to validate the supplied credentials. However, if we want to customize the authentication logic used by the Login control, we can create our own event handler for this event. We just need to be certain to set the `e.Authenticated` property within the event handler to True if the user's credentials are valid, False otherwise.

For this example I decided to augment the authentication logic to authenticate on not only the user's username and password, but also on their existing email address and successfully responding to a CAPTCHA. A CAPTCHA is a technique for ensuring that a human is filling out a web form (and not some automated robot program); CAPTCHAs are commonly implemented as an image that displays some distorted text and asks the user to enter the text in the image. The idea here being that a computer program will be unable to analyze the image and decipher the inner text, whereas a human can do this easily and quickly. For the CAPTCHA piece I am using Jeff Atwood's free and excellent ASP.NET CAPTCHA server control.

To create this custom authentication logic example, I started by converting a standard Login control into a template and added a TextBox named `Email` (with a corresponding RequiredFieldValidator) along with Jeff's CAPTCHA control:

```
<asp:Login ID="Login1" ...>
    <LayoutTemplate>
        <table border="0" cellpadding="1" cellspacing="0" style="border-collapse:
```

```
collapse">
            <tr>
                <td align="center" rowspan="5" style="padding:15px; font-weight: bold;
color: white; background-color: #6b696b">
                   Log In
                </td>
                <td style="width:8px;"> </td>
                <td align="right">
                   <asp:Label Font-Bold="True" ID="UserNameLabel" runat="server"
AssociatedControlID="UserName">User Name:</asp:Label></td>
                <td>
                   <asp:TextBox ID="UserName" runat="server" TabIndex="1"></asp:TextBox>
                   <asp:RequiredFieldValidator ID="UserNameRequired" runat="server"
ControlToValidate="UserName"
                        ErrorMessage="User Name is required." ToolTip="User Name is
required." ValidationGroup="Login1">*</asp:RequiredFieldValidator>
                </td>
                <td align="center" rowspan="5" style="padding:15px;">
                    <asp:Button ID="LoginButton" runat="server" CommandName="Login"
Text="Log In" ValidationGroup="Login1" TabIndex="5" />
                </td>
            </tr>
            <tr>
                <td style="width:8px;"> </td>
                <td align="right">
                    <asp:Label Font-Bold="True" ID="PasswordLabel" runat="server"
AssociatedControlID="Password">Password:</asp:Label></td>
                <td>
                    <asp:TextBox ID="Password" runat="server" TextMode="Password"
TabIndex="2"></asp:TextBox>
                    <asp:RequiredFieldValidator ID="PasswordRequired" runat="server"
ControlToValidate="Password"
                         ErrorMessage="Password is required." ToolTip="Password is
required." ValidationGroup="Login1">*</asp:RequiredFieldValidator>
                </td>
            </tr>
            <tr>
                <td style="width:8px;"> </td>
                <td align="right">
                    <asp:Label Font-Bold="True" ID="EmailLabel" runat="server"
AssociatedControlID="Email">Email:</asp:Label></td>
                <td>
                    <asp:TextBox ID="Email" runat="server" TabIndex="3"></asp:TextBox>
                    <asp:RequiredFieldValidator ID="EmailRequired" runat="server"
ControlToValidate="Email"
                        ErrorMessage="Email is required." ToolTip="Email is required."
ValidationGroup="Login1">*</asp:RequiredFieldValidator>
                </td>
            </tr>
            <tr>
                <td style="width:8px;"> </td>
                <td align="right">
                    <b>Verification:</b>
                </td>
                <td>
                    <cc1:CaptchaControl id="CAPTCHA" runat="server"
ShowSubmitButton="False" TabIndex="3" LayoutStyle="Vertical"></cc1:CaptchaControl>
                </td>
            </tr>
```

```
            <tr>
                <td style="width:8px;"> </td>
                <td colspan="2">
                    <asp:CheckBox ID="RememberMe" runat="server" Text="Remember me next
time." TabIndex="4" />
                </td>
            </tr>
        </table>
        <asp:ValidationSummary ID="ValidationSummary1" runat="server"
ShowMessageBox="True" ValidationGroup="Login1"
            ShowSummary="False" />
    </LayoutTemplate>
</asp:Login>
```

Next, I created an event handler for the `Authenticate` event. Here, I start by checking to determine if the CAPTCHA is valid. If it is, I then use the `Membership.ValidateUser` method to ensure that the user's username and password are valid. Assuming that they are, I next get their email on file by using `Membership.GetUser(`*`username`*`)` and comparing the `Email` property to the email supplied by the user. If any of these checks fail, I set `e.Authenticated` to False; if they all pass, I set it to True. I update the Login control's `FailureText` property depending on the check that fails to provide a more descriptive explanation for the reason they were unable to login.

Note that to access controls (such as the `Email` TextBox or CAPTCHA control) from the `<LayoutTemplate>` we use `LoginControl.FindControl("ID")`.

```
Protected Sub Login1_Authenticate(ByVal sender As Object, ByVal e As
System.Web.UI.WebControls.AuthenticateEventArgs) Handles Login1.Authenticate
    'We need to determine if the user is authenticated and set e.Authenticated
accordingly
    'Get the values entered by the user
    Dim loginUsername As String = Login1.UserName
    Dim loginPassword As String = Login1.Password

    'To get a custom Web control, must use FindControl
    Dim loginEmail As String = CType(Login1.FindControl("Email"), TextBox).Text

    Dim loginCAPTCHA As WebControlCaptcha.CaptchaControl =
CType(Login1.FindControl("CAPTCHA"), WebControlCaptcha.CaptchaControl)

    'First, check if CAPTCHA matches up
    If Not loginCAPTCHA.UserValidated Then
        'CAPTCHA invalid
        Login1.FailureText = "The code you entered did not match up with the image
provided; please try again with this new image."
        e.Authenticated = False
    Else
        'Next, determine if the user's username/password are valid
        If Membership.ValidateUser(loginUsername, loginPassword) Then
            'See if email is valid
            Dim userInfo As MembershipUser = Membership.GetUser(loginUsername)
            If String.Compare(userInfo.Email, loginEmail, True) <> 0 Then
                'Email doesn't match up
                e.Authenticated = False
                Login1.FailureText = "The email address you provided is not the email
address on file."
            Else
                'Only set e.Authenticated to True if ALL checks pass
                e.Authenticated = True
            End If
```

```
        Else
            e.Authenticated = False
            Login1.FailureText = "Your username and/or password are invalid."
        End If
    End If
End Sub
```

Checking for CAPTCHA Validation First

Initially, my code performed the check for membership validity first, then the email check, and then the CAPTCHA check. However, after a discussion of this approach on my blog, I decided to change the order such that the the CAPTCHA was performed first due to the following comments:

Alert reader Israel A. notes: "Wouldn't be more interesting to validate captcha code before Membership user? If captcha code is invalid, you don't access the membership methods (like ValidateUser) and, in this case, can cause an extra overhead."

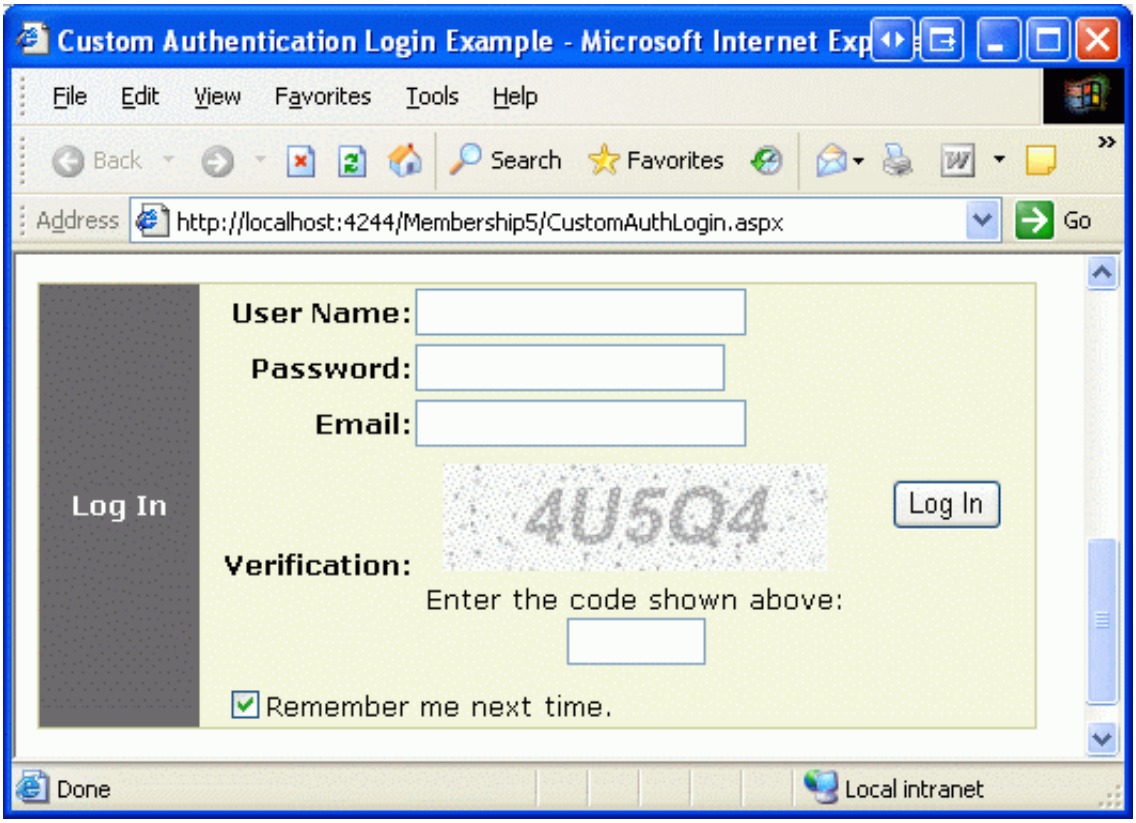
It may be more efficient to first check for CAPTCHA validity before checking the membership validity, since examining membership information for the `SqlMembershipProvider` requires a request to the database. Something to consider, although the performance gain is likely very negligible since logging in is a rather infrequent occurrence and logging in with an invalid CAPTCHA is even less likely. But if you want to cut as much fat from the system as possible, consider reworking the event handler to follow Israel's suggestion.

nstlgc follows up with the following comment: "IMHO the fact that you process your CAPTCHA only after a succesful login defies the whole purpose of having a CAPTCHA to begin with. CAPTCHA serves to verify that the entity visiting the website is not a robot. The prominent reason for having a robot visit your site is likely to be password bruteforcing. Your solution will happily allow robots to try until they find a login/password pair that works, then report to their owner for actual CAPTCHA clearance."

nstlgc makes a good point, especially since the output for an invalid CAPTCHA differs from the output if the membership validation fails. Finally, James B. chimes in with a comment on CAPTCHAs in general and web accessibility: "The use of CAPTCHA makes many websites illegal due to accessibility regulations , (not just Section 508, but worldwide). Its use certainly frowned upon for public sites where the abilities of users cannot be guaranteed. Perhaps a warning to the usage of CAPTCHA might be appropriate. Of course the problem is for blind or visually impaired users, they have no idea what the caption says. For completeness on this subject, check out http://www.w3.org/TR/turingtest/"

Follow the discussion at this blog entry...

With these changes, the Login control now includes a prompt for the user's email and a CAPTCHA, as the following screen shot shows, and only allows the user to log in if they provide their correct username, password, email address, *and* satisfy the CAPTCHA.



Conclusion

In this article we saw how to customize the appearance and layout of Login control both by configuring its properties and by converting the Login control into a template. By converting the Login control into a template, we can add additional Web controls, such as a CAPTCHA. By creating an event handler for the Login control's `Authenticate` event, we are then able to use these additional Web controls to augment the authentication logic.

The Login control is a great example of ASP.NET 2.0's ability to make the simple case easy while still allowing for more customized solutions without too great an effort. In the simple case, you can just drag and drop a Login control from the Toolbox onto your Login page. That's it, you're done! However, with templates and its myriad of events, the Login control can be extended to include alternative interfaces and authentication logic.

Happy Programming!

- By Scott Mitchell

Attachments

- Download the code used in this article

Article Information	
Article Title:	ASP.NET.Examining ASP.NET's Membership, Roles, and Profile - Part 5
Article Author:	Scott Mitchell
Published Date:	June 7, 2006
Article URL:	http://www.4GuysFromRolla.com/articles/060706-1.aspx